Evolutionary Rapid Development

SPC-97057-CMC

Version 01.00.04

June 1997

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM
SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

Evolutionary Rapid Development

Richard M. De Santis, John Blyskal, Assad Moini, Mark Tappan

A description of the process used to support the evolutionary development of advanced information retrieval systems.

It is change, continuing change, inevitable change, that is the dominant factor in society today. No sensible decision can be made any longer without taking into account not only the world as it is, but the world as it will be. . . . This, in turn, means that our statesmen, our businessmen, our everyman must take on a science fictional way of thinking.

Isaac Asimov (1920-1992),

Russian-born U.S. author. "My Own View"
The Encyclopedia of Science Fiction,
ed. by Robert Holdstock, 1978; repr. in
Asimov on Science Fiction, 1981.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1997	Technical Report - Final

4. TITLE AND SUBTITLE

Evolutionary Rapid Development

5. FUNDING NUMBERS

G MDA972-96-1-0005

6. AUTHOR(S)

Produced by Software Productivity Consortium under contract to Defense Advanced Research Projects Agency

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES)

Software Productivity Consortium
SPC Building
2214 Rock Hill Road
Herndon, VA 20170

8. PERFORMING ORGANIZATION REPORT NUMBER

SPC-97057-CMC,
Version 01.00.04

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA/ITO
3701 N. Fairfax Dr.
Arlington, VA 22203

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION / AVAILABILITY STATEMENT

No Restrictions

12b. DISTRIBUTION CODE

1

13. ABSTRACT *(Maximum 200 words)*

This technical report provides an overview of the use and application of the Evolutionary Rapid Development process. The Evolutionary Rapid Development (ERD) process helps manage the development of complex systems in an environment of rapidly evolving components and architectures.

Fundamental to ERD is the concept of composing software systems based on the reuse of components, the use of software templates and on an architectural template. Continuous evolution of system capabilities in rapid response to changing user needs and technology is highlighted by the evolvable architecture, representing a class of solutions. The process focuses on the use of small artisan-based teams integrating software and systems engineering disciplines working multiple, often parallel short-duration time-boxes with frequent customer interaction.

Key to the success of the ERD-based projects is parallel exploratory analysis and development of features, infrastructures, and components with and adoption of leading edge technologies enabling the quick reaction to changes in technologies, the marketplace, or customer requirements.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Evolutionary Development, Rapid Development, Software Process, Software Exploration	28
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

1.0 Introduction

The Software Productivity Consortium is tasked as a technology development and integration agent for the Information Technology Office (ITO) of the U.S. Defense Advanced Research Projects Agency (DARPA). The Consortium provides DARPA ITO with rapid application development and integration resources to assist in the definition, development, deployment, and operation of DARPA ITO Intranet resources. The Consortium provides operational components in the form of information repositories, knowledge bases, tools, software components, and hardware architectures to support experimental integration of ITO technological assets. As part of its continuing support for ITO, the Consortium has developed the following technologies:

- CYBERosetta. An information repository and its associated tools to collect, categorize, and provide World Wide Web-based access to technical information artifacts relevant to DARPA. It is the basis for the experimental integration of DARPA ITO research and developmental components.

- Prototype System of System/Seamless Link to Information (PSOS/SLI). A distributed processing, middleware, and repository infrastructure that draws state and knowledge from the products communicated by multiple interconnected heterogeneous systems and data sources

To guide these research and development efforts, the Consortium is developing the Evolutionary Rapid Development (ERD) process to help manage the development of complex systems in an environment of rapidly evolving components and architectures. Contemporary process models generally lack sufficient guidance to address component-based rapid integration, delivery, and maturation strategies. These models tend to assume you are building a one-of-a-kind system and consider the development and maintenance life cycles as separate events.

Fundamental to ERD is the concept of composing software systems based on the reuse of components, the use of software templates and on an architectural template. Continuous evolution of system capabilities in rapid response to changing user needs and technology is highlighted by the evolvable architecture, representing a class of solutions. The process focuses on the use of small artisan-based teams integrating software and systems engineering disciplines working multiple, often parallel short-duration timeboxes with frequent customer interaction.

Key to the success of the ERD-based projects is parallel exploratory analysis and development of features, infrastructures, and components with and adoption of leading edge technologies enabling the quick reaction to changes in technologies, the marketplace, or customer requirements.

The ERD process enables the Consortium to rapidly deliver working versions of systems for:

- Exploratory development - where the concepts are fuzzy and must be evaluated

- Proof-of-concept development - where concepts, once articulated, can be realized and their usefulness evaluated

- Pilot projects - where concepts, once proven, can be implemented in systems for use in controlled circumstances

The ERD process was originally developed to support research and development efforts and to serve as a general guide for producing working prototypes for evaluating and exploring concepts within CYBERosetta and PSOS/SLI. During this period, ERD was first employed in a high risk, high payoff research domain where exploration and experimentation were fundamental and learning from failures was as valuable as learning from successes. We performed the ERD activities in an informal manner with more emphasis in some areas than others. Recently we have begun to use the ERD process in the production of operationally deployed systems for use within the Consortium and for use by our clients. As we gain more experience with the process, we continue to refine and improve its definition and continue to formalize its execution.

This document provides an overall description of the ERD process and the motivation behind it, as well as lessons learned from applying it within our projects. The document is organized into the following sections:

- 1.0 Introduction. This section introduces the ERD process and presents the organization of this report.
    - 1.1 Motivation. This section describes the major factors that influenced our design of the ERD process. It describes external business and technological concerns that the process must address for us to develop systems consistently that meet our customers' needs.
    - 1.2 Key Features. This section provides an overview of the major concepts and characteristics embedded in the process.
- 2.0 Evolutionary Rapid Development Process Description. This section provides an overview of the ERD process roles and the major activity categories:
    - Concept Initialization/Resetting activities
    - Evolutionary Development Time Box activities
    - Customer and End-User Operational Leverage Evaluation activities.
- 3.0 Experiences. This section describes some of the Consortium's experiences in implementing and applying the process.
- 4.0 Next Steps. This section describes some of the areas the Consortium intends to pursue as we continue to refine and improve the process.

1.1 Motivation

Information systems development is undergoing a revolution. Developers no longer focus on handcrafting an application; rather they assemble robust, fully functional systems from readily available and inexpensive components. The system developer's task has begun to transform itself

from one of specifying, designing, and implementing custom solutions to one of evaluating, selecting, tailoring, and integrating hardware and software components in accordance with an application framework [Booch 1997]. We have entered the "Information Age" [Lewis 1995], where change is constant and we live in "real time." Computer literacy is becoming the norm, and end users are beginning to define their own applications [Jones 1995]. The Gartner Group predicts that by 1999, at least 70 % of applications will be built in business units by either decentralized applications development staff or power users.

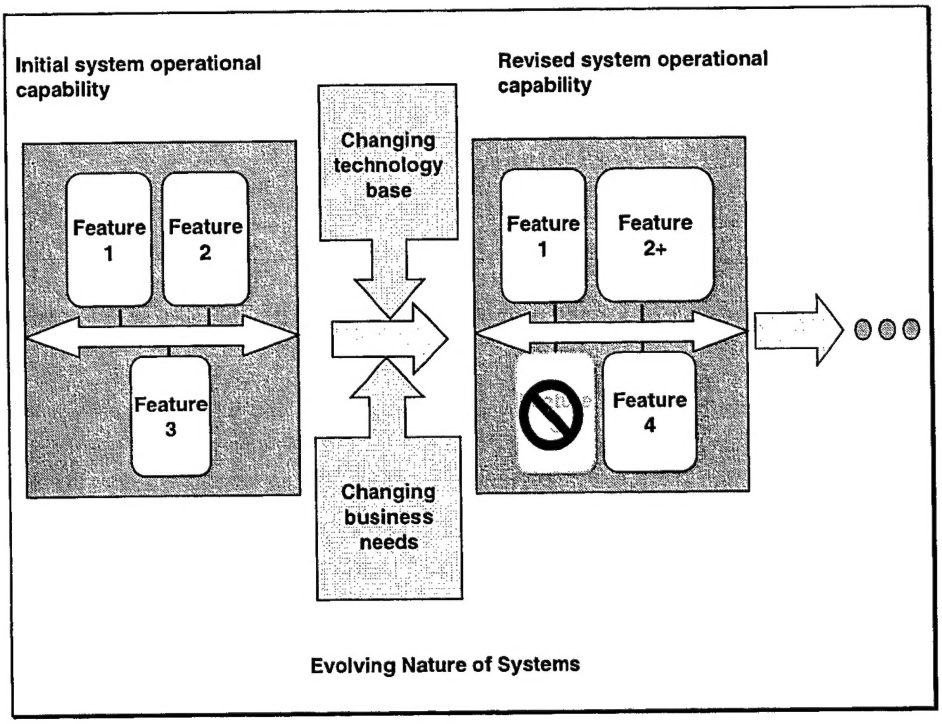

Figure 1. Evolving Nature of Systems

The dynamics of the systems development/usage environment make it difficult to specify and define a system in full detail before it is fielded. As illustrated in Figure 1, systems must be designed to grow and adapt to future events that are not always certain. Features may be added, revised, or removed based on both user needs and changes to the way technology is used (e.g., emergence of intranets). Observations about the system/software and business environments that significantly influenced the design of the ERD process are described below:

- Time to market

  A product that is 5 months late to market can cost one third of its profits [Parasoft Corporation April 1996]. More importantly, not only is first to market important, but first to *market acceptance* is critical. This factor makes it essential that user needs be identified and satisfied quickly. You must consider the compatibility of your products with previous versions, as well as the inherent skills and knowledge of your users, assuring that the users will understand and leverage the capabilities of your system.

- Continuous product evolution and enhancement

For a system to be useful, it must evolve through use in its intended operational environment. A product is never "done;" it is always maturing as the usage environment changes. As Figure 2 demonstrates, we often try to define a system using our most familiar frame of reference — where we are now. We make assumptions about the way business will be conducted and the technology base on which the business will be implemented. A plan is enacted to develop the capability, and, sooner or later, something resembling the envisioned system is delivered.

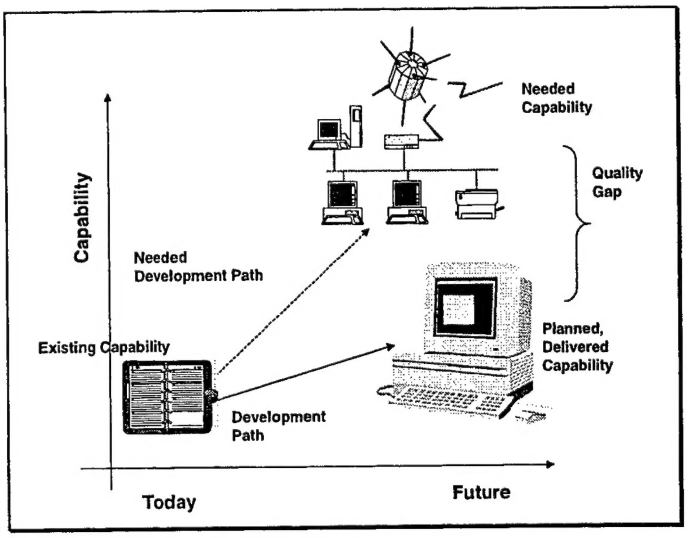

Figure 2. The Quality Gap.

Unfortunately, that capability, although it is what the user has asked for, is not what the user then needs. Often the environment in which the system is to be used is not what we originally expected. This is the quality gap — getting what you asked for, but not what you truly need. The problem is exacerbated by allowing significant time to elapse between the time a need is identified and when the capability is delivered. Therefore, it is necessary to include frequent user feedback and to assess the operational value of the system. The process must plan for change in a controlled manner.

- Rapidly changing world of COTS — software as a commodity

Functionality that not too long ago would have required significant investment to create can now be bought off-the-shelf for a fraction of its development cost. The infrastructure to help standardize how software components can interoperate is now reaching maturity — such as Microsoft's Distributed Component Object Model (DCOM) and the Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA). This development has spawned a component-oriented marketplace. Systems developers can now leverage both applications and pre-packaged components acquired from various sources to deliver functionality quickly.

Hardware, software, and network advances

The industrial revolution enhanced muscle power with machinery; the computer revolution is enhancing the power of the human brain. Moore's Law, originally formulated by Gordon Moore in the early 1960's and later revised, observes that the

power and complexity of the silicon chip doubles approximately every 18 months, see Figure 3. The effect of this growth is that roughly every 18 months a new chip becomes available that shrinks in size, doubles in power, and drops in price (e.g., prices of less than a dollar a megahertz.) This growth is forecast to continue, constrained only by the laws of physics.

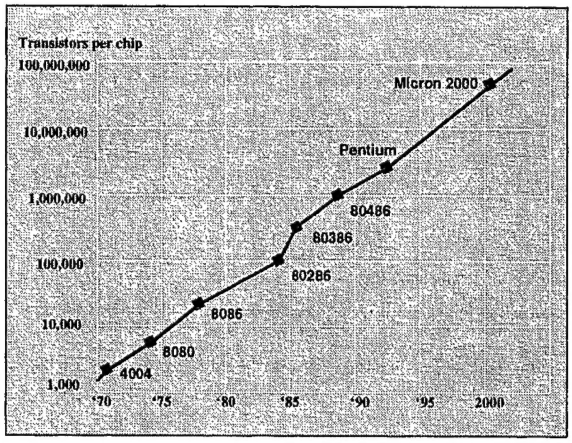

Figure 3. Moore's Law

The fiber optic and wireless communication revolution both feeds off of and fans the hardware/software flames. A 60 % to 90 % drop in the cost of wireless telephony over the next few years is projected [From Wires to Waves]. This will fuel growth in the use of the Internet, and with it increase the need for groupware and collaboration technologies in a distributed computing environment. This includes distributed (or virtual) software development teams.

The rapid growth we are experiencing affects how we view the design and development of systems. No longer can we afford to plod along, procure target computers, spend months or years developing, and then field obsolete solutions. The developer's success is linked to a wider environment of interest: the emerging technologies and changing business landscape. The developers, and the products they produce, must be agile and swift to intersect these technologies, stimulate user/buyer interest, and deliver capability in a cost-effective and timely manner.

- Adaptable Architectures and Application Program Interfaces

To have flexibility, you must have a system that can tolerate change. A system will remain viable only if it can evolve to serve new needs. If it cannot adapt, it will become extinct. This places important conditions on overall system design. It must be able not only to accommodate unanticipated changes to users' needs (i.e., functional), but also to changes in the underlying technology used to implement the system. The system's

developer must consider not only a solution for a specific customer, the developer must consider a family of solutions for a set of customers — if they expect to leverage their efforts and stay competitive. A robust, loosely coupled, highly cohesive architecture is essential. You must be able to substitute components from different sources and allow them to interact across platforms. This objective requires careful partitioning and encapsulation of services with application program interfaces, adherence to existing and future standards, and incorporating vendor frameworks as necessary (e.g., Microsoft Foundation Class). A modular structure promotes productivity in the development team, allowing for features to be developed in parallel, and helps to minimize the need for frequent coordination across the entire team.

- Importance of information retrieval and knowledge representation technologies

  The Internet/intranet are becoming the primary means for us to obtain information. With the ability to publish documents instantaneously and to make them available throughout the world, we are reaching information overload. Alta Vista (May 1996) had an index of more than 30 million pages, which received 22 million hits per day in November 1996. Their spider collects more than 3 million Web pages a day, with an overall index of more than 40 gigabytes.

  The information is there, but leveraging it is another matter. Finding information relevant to a specific information need is difficult and time consuming. Simple search requests result in an avalanche of returned objects. For example, a search for "software architectures, templates, or patterns" can yield information from many domains and overwhelm the searcher (i.e., items dealing with PowerPoint presentation "templates"). Current search engine interfaces provide only rudimentary help in down-selecting the most relevant information, requiring search users to build elaborate descriptions of their information needs.

  With this "everyone can publish" ability, the cataloging techniques of the past are stressed. Information retrieval, knowledge representations, and knowledge management techniques are beginning to address these issues by providing intelligent search, browse, learning, and knowledge sharing agents [Lewis 1995]. These technologies will be pivotal to any modern information system. The market for intranet search and retrieval tools is expected to grow five fold, from $53 million to $255 million over the next few years (Zona Research1996.)

- Acquiring and sustaining a talented development staff

  Technology changes fast. Finding and keeping the right talent pool conversant in this technology is critical. Flexibility is required to acquire and utilize people who possess a deep understanding of both the technology and the way to implement systems using it. They must be creative self-starters who are highly motivated, challenged, flexible, and able to grasp new technologies quickly.

  Downsizing pressure (AMA Survey 1996) affects staffing profiles by reducing the

number of full time personnel available and focusing skill sets on emphasizing technical know how. You need to maintain a core competency in the systems you build, yet quickly augment it with individuals from outside the organization when necessary. This puts demands on the way you organize a development team. People must become productive without climbing a steep learning curve. You cannot rely on the availability of the original developers once the system is fielded.

Maintaining the core development staff requires an emphasis on training and retaining technology-oriented management and support teams throughout the organization. They must understand the concepts of the current/future technology and be able to relate to and interact effectively with the core developers.

- Exploratory nature of research and development

  The research and initial product development arena is geared to high-risk, high-payback efforts. You must plan for setbacks and take into account that during development there are times that obtaining reliable information is more important than producing a product. Quite often a development goal is to explore or provide proof-of-concept experience. The ability to identify and eliminate unproductive alternatives quickly is necessary to assure that development proceeds on a stable foundation. You don't necessarily measure progress against what is built, but rather against what is now known and the risk that has been reduced.

  The need to obtain better and more reliable information sometimes causes tension when the ultimate goal is to develop and field a product to meet an immediate need. The development process must explicitly support integrating all life-cycle viewpoints into a cohesive understanding not only of end-user functionality, but also system infrastructure. All stakeholder viewpoints must be represented throughout the development effort.

- Continuous process improvement

  Processes guide teams to meet their objectives, such as delivering solutions to user needs, effectively and efficiently. In a dynamic, changing environment, processes must be flexible to handle these changes.

  The process should enable communication and coordination within the teams, assure good visibility into progress, and foster creativity and productivity. The process should be ubiquitous and reflect the way work is actually conducted. . This means that the process must be easily maintained and constantly improved as the organization learns and adapts.

1.2 Key Features

In Section 1.1 we discuss some of the factors of the software/systems/business environment that influence the definition and character of our development process. For the ERD process we established a set of process characteristics that together address these factors. The process is focused on providing the appropriate level of control, while stimulating creativity and innovation, and delivering products quickly. Some key features of the ERD process are described

below:

- Significant customer/end-user participation.

  CYBERosetta and PSOS/SLI systems represent new paradigms for gathering, displaying, organizing, and accessing information. As with all innovations, users cannot adequately articulate their information needs and system requirements early in the life cycle. The vision for the system and how it could be used must evolve along with the users' understanding of their needs. As the system gains capability, the users become aware of additional needs and set updated priorities for the features to include in the system.

  Customers, who are concerned primarily with business issues, such as leveraging business operations, system life-cycle costs, and integrating the system into their existing infrastructure, provide a strategic, life cycle-oriented viewpoint. Customers help establish the business objectives and design constraints for the system.

  The end-user perspective represents the functional user. End users are the source of system features; they also provide insight into the design and implementation details for the human-computer interface. They help identify the types of information and associated meta-data that the system should store.

  To elicit customer/user input, we hold frequent scheduled and ad hoc/impromptu meetings with the stakeholders. Demonstrations of system capabilities are held to solicit feedback before design/implementation decisions are solidified. Frequent releases (e.g., beta) are made available for use to provide insight into how the system could better support user and customer needs. This assures that the system evolves to satisfy existing user needs.

- "Time boxing"

  The ERD process is structured to use demonstrated functionality rather than paper products as a way for stakeholders to communicate their needs and expectations. Central to this goal of rapid delivery is the use of the "time box" method [McConnel 1996]. Time boxes are fixed periods of time in which specific tasks (e.g., developing a set of functionality) *must* be performed. Rather than allowing time to expand to satisfy some vague set of goals, the time is fixed (both in terms of calendar weeks and person-hours), and a set of goals is defined that realistically can be achieved within these constraints. The iterative use of short-duration, often parallel, time boxes and frequent customer interaction helps to:

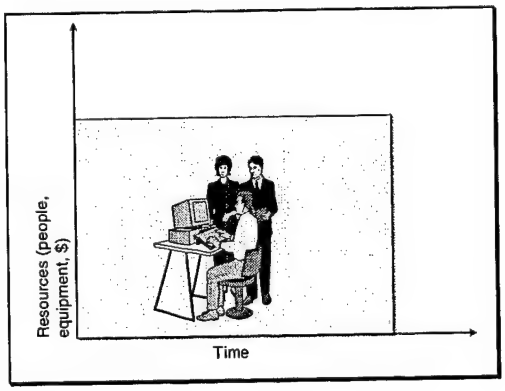

Figure 4. Time Box

- keep pace with the rapidly changing technology marketplace
- minimize the amount of "throw away" software that is developed over the lifespan of the project
- manage the volatility of COTS components.

Time box goals may be anything from providing information on design tradeoffs and testing concepts to delivering functionality to the end users. Fixing specific goals has the benefit of forcing the stakeholders (e.g., the customer and developer staff) to rank and be specific about the goals, and to discard low-priority items. This technique assures that something is delivered and that the problems are reduced to their simplest form. Teams can concentrate on cooperation and creativity to meet the time constraints. It also assures that for the duration of the time box, key personnel and resources (e.g., computing, network platforms) are fully committed to the goals of the time box, and there is less chance that they can be sidetracked onto lower-priority work. An overall concept for the system is used to guide each time box. Results (product or information) from one time box are the basis for planing and executing another.

- COTS-based, reuse driven approach.

  The ERD process is based on an aggressive strategy that systematically identifies and uses government-, industry-, or academia-provided components, tools, and technology as a basis for composing the system. The development environment supports the use of domain engineering and component-ware approaches. It relies on a close integration of systems and software engineering disciplines to identify and make tradeoffs between cost/schedule functionality and risk in using existing components or vendor-supplied integration infrastructures (e.g., DCOM versus CORBA). The COTS-based approach for incorporating reuse into the development process leverages the use of templates and frameworks whenever possible; and in some cases may require that new templates be synthesized from existing components or example software.

- Small team composition.

  The most important element of a successful system development effort is people. It is a significant challenge to establish organizational structures and roles to ensure effective assignment of responsibilities and enhance communication. The ERD process invests heavily in maintaining its human resources. Highly experienced individuals are cloistered together and provided liberal amounts of computer infrastructure, tooling, and access to technical literature and knowledge bases. The process includes a set of roles defining responsibilities within the development team. The roles establish a career path whereby junior individuals can participate in highly specialized programming roles and learn from more senior members through activities such as code reviews and testing using a test buddy approach.

- Continuous process and product improvement.

The most effective strategy to address the quality gap is that of evolutionary development. Use of this technique shortens the time between specifying a need and delivering a solution. Since it is an iterative strategy, evolutionary development allows for re-specifying the need and the technology as the user, business, and technology environments change. Evolutionary delivery focuses on delivering the 80% of the capability that has immediate value in 20% of the time, rather than focusing on the 20% of the capability that would take more than 80% of the time. Unproductive iteration is controlled by an overriding vision, or concept, for the product and its use. Each iteration either further refines the vision or incrementally produces versions to satisfy all or part of the vision. Feedback between increments provides for both adjusting the vision and determining the features to implement across each iteration.

The team is constantly on the lookout for ways to improve the products we deliver and the process we use. Team members typically spend 10% of their time investigating and maintaining currency in technologies or products. Promising technology or components are often evaluated through trial usage. Lessons learned from using the process are fed back into the process description maintained by the librarian and process manager.

- Cusp identification

  Analysis and planning for change is central to the ERD process. Cusp identification deals with identifying trends in the emergence and use of technology within an industry. This practice requires tracking the publishing and acceptance of standards, identifying changes in the marketplace for price/performance, observing the number of applications using a technology, and identifying trends within industry surveys. The objective is to position the development (e.g., the architecture) to anticipate and react to beneficial changes at the right time.

- Architecture-based composition and integrated domain engineering.

  The design framework for the system is based on using existing published or de facto standards. The system is organized to allow for evolving a set of capabilities that includes considerations for performance, capacities, and functionality. The architecture is defined in terms of abstract interfaces that encapsulate the services and their implementation (e.g., COTS applications). The architecture serves as a template to be used for guiding development of more than a single instance of the system. It allows for multiple application components to be used to implement the services. A core set of functionality not likely to change is also identified and established.

- Long-range/short-range planning.

  Evolutionary development faces the risk of being "short sighted," reacting to individual user needs while losing the ability to define and achieve an overall vision for the system. To keep development from degenerating into a "random walk," long-range plans are defined to guide the iterations. These plans provide a vision for the overall system and set boundaries (e.g., constraints) for the project. Each iteration within the process is

conducted in the context of these long-range plans. The ERD process assumes that the system will never cease to exist, but will be transformed and adapted as it is used. Therefore, the time period that the long-range plan covers is extended as necessary. There is a continuum from concept through composition and deployment/enhancement. Decisions about the process or product are made from a life-cycle perspective.

- Continuous integration and testing.

  A tenet of the ERD process is that the most reliable measure of progress is through execution of the system. As part of the ERD process, once an architecture is established, software is integrated and tested on a daily basis. This allows the team to assess progress objectively and identify potential problems quickly. If a build "broke" (i.e., failed to compile/link or execute some basic functionality), this would be known immediately. Since small amounts of the system are integrated at one time, diagnosing and removing the defect is rapid. User demonstrations can be held at short notice since the system is generally ready to exercise at all times.

  This integrate/test discipline requires developers to organize and rank their delivery of features carefully to ensure they can be absorbed at the right time and in harmony with other features. This helps enforce modularity in the system implementation, requiring the implementation to be loosely coupled.

2.0 Evolutionary Rapid Development Process Description

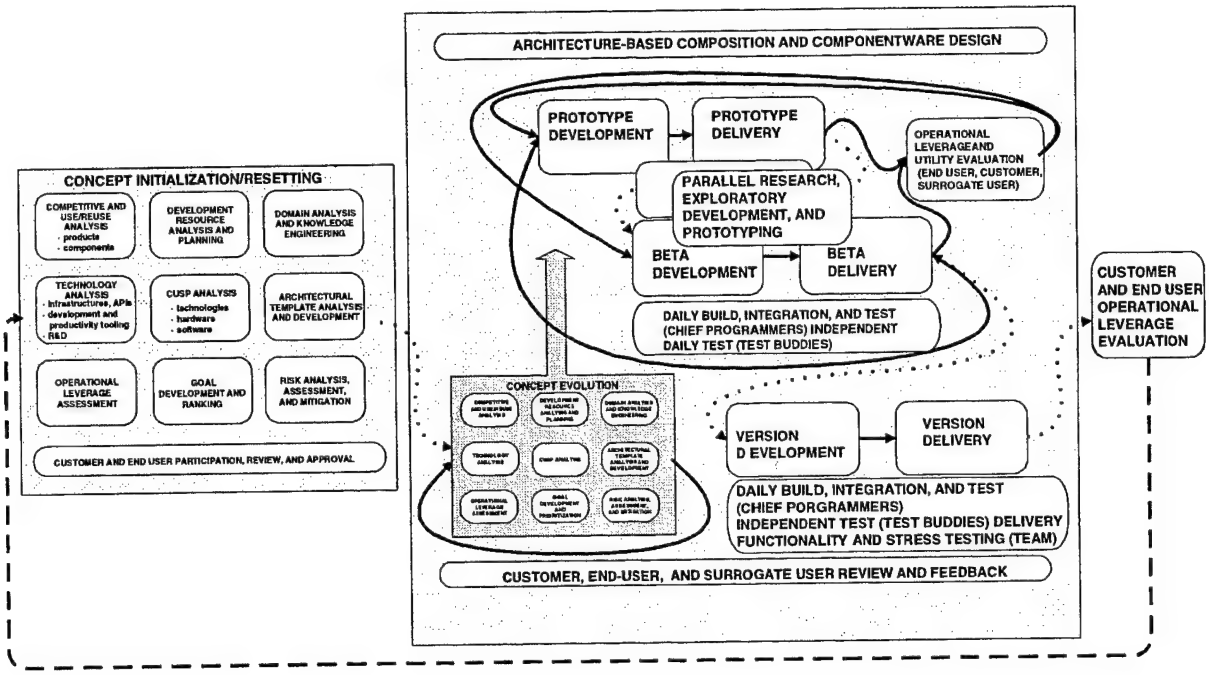

Figure 5. Evolutionary Rapid Development Process

The ERD process consists of three main parts, as depicted in Figure 5. The first part, the Concept Initialization and Resetting activities, establishes the high-level objectives and constraints for the system. This is where the long-range objectives (e.g., requirements) for the system are established, alternatives that will be explored are identified, and the short-term goals for the development increments are agreed upon. The second part, the Evolutionary Development Time Box activities, refines and implements the system through a series of prototypes. The third part, the End-User Operational Leverage Evaluation activity, assesses the operational system to determine its value and to identify additional needs. This operational assessment cycles information back into the Concept Initializing/Resetting activities either to refine or to significantly evolve the concept. This evaluation and setting of new needs supports the continued growth of the system to meet business needs.

The roles and the three major activities of the ERD process are described in this section.

2.1 Process Roles

Individuals participate in the ERD process in one or more of the roles shown in Figure 6. These roles represent specialization of an individual's skills, knowledge, and responsibilities within the process. An individual typically fulfills more than one role within the project, and not all roles are staffed throughout the project life cycle.

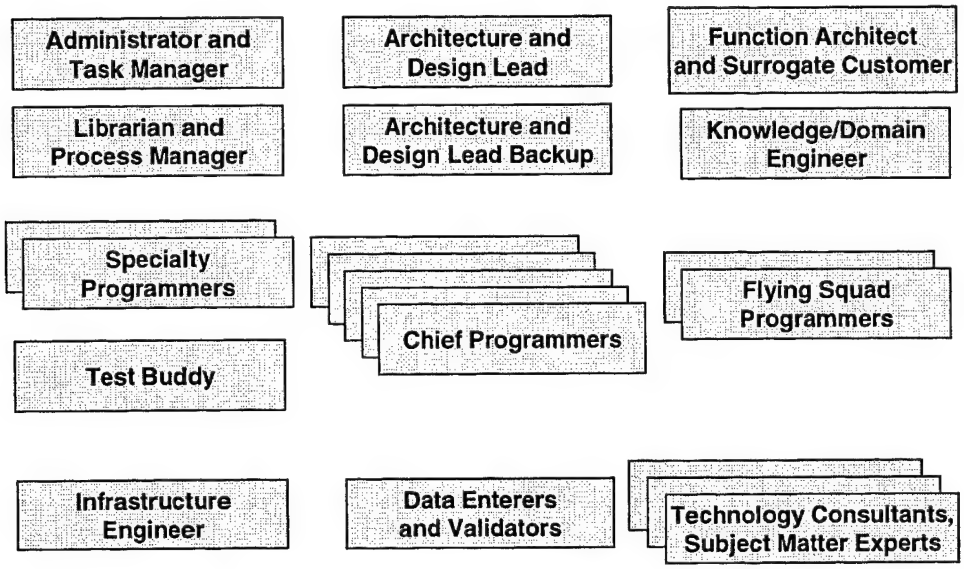

Figure 6. ERD Process Roles

The three top roles—the administrator and task manager, the architect and design lead, and the functional architect and surrogate customer—coordinate closely to balance the project resource, technical, and customer/user business needs. The architect and design lead (including backup) and the chief programmers represent the central technical core for the development. The infrastructure engineer, data entry and validation, and technology consultants/subject matter expert roles are typically part-time, filled on an as-needed level-of-effort basis. The librarian and process manager, test buddy, chief programmer, and flying squad programmer roles are periodically rotated among individuals, while the other roles typically are assigned for the duration of a project.

The ERD process roles are described below:

- Function architect and surrogate customer.

  This role requires understanding the business problem and the usage environment being addressed, and acting as the interface and ombudsman for the end users and customers.

The functional architect responds to user needs that typically are defined in terms of system features. The surrogate customer responds to business needs, such as continual re-evaluation of operational leverage, cost/schedule, and integration issues. The functional architect and surrogate customer roles may be fulfilled by one or more individuals who closely coordinate their efforts.

- Administrator/task manager.

  This role extends the traditional project management monitoring functions (e.g., tracking cost, schedule, quality) to one of facilitation. Examples of this individual's responsibilities include handling logistics, such as procurement and software license issues, and managing external interrupts that may divert team member resources. This role can be shared by one or more individuals.

- Architect and design lead.

  The ERD process uses an architecture-driven development approach that makes this role critical to the success of the project. The significant developmental issues and greatest sources of risk tend to be not in the detailed design or implementation of the software code, but in establishing a robust and efficient architecture to serve as a framework for development. This requires specific knowledge and skills separate from those of a programmer. The architect and design lead has the responsibility to define and evolve the architecture, and to oversee its implementation. This responsibility also includes participation in reviewing any code and test status.

- Architect and design lead (backup).

  This individual is an additional resource for the architect and design lead, providing aid and backup. The individual in this role is kept cognizant of all architectural decisions and assumes some of the architect and design lead's duties when necessary. Usually this occurs when multiple time boxes are active, such as when planning for the next phase. This individual may also lead explorations into architectural tradeoffs to provide information to the architect and design lead. This individual provides daily direction to the flying squad programmers in reviewing code and monitoring tests.

- Librarian and process manager.

  This person is responsible for maintaining all project documentation and managing changes to the process. The librarian is responsible for the project developmental and delivery libraries (e.g., Configuration Management baselines). This includes all product documentation (such as reference materials and vendor products) and descriptions of interfaces, protocols, and design notes. The librarian acts as the lead for negotiating software interfaces with groups external to the project. The process manager responsibilities deal with assuring that established processes are followed faithfully, as well as updating the process documentation to reflect changes.

- Chief programmer.

Chief programmers are senior technical persons who are responsible for realizing the system architecture and responsible for implementing the infrastructure and components necessary to populate the architecture. These persons have a thorough understanding of the components and how to adapt them. A chief programmer may be assigned to implement a specific part of the architecture, such as the user interface or search services. Chief programmers produce the "glue" necessary to interface components together.

- Specialty programmer.

  These individuals support the chief programmers with technical expertise in one or more programming domains. Typical specialties are authoring help/user documentation, graphics, or database administration. Specialty programmers also implement components that are required by the project but are not available off-the-shelf. These roles are staffed as specific skills are identified and are obtained either through a matrix structure from the organization or by contract from outside.

- Flying squad programmers.

  These individuals represent a programming pool used to support the chief programmers and backup architect. When not providing additional programming resources, they set up and conduct demonstrations of prototypes or install software in the user environment as needed. The also work on risk mitigation and do exploratory (throw-away) development to provide insight into development alternatives.

- Test buddy.

  These individuals work side by side with programmers to test code continuously as it is developed. Their responsibilities are not only to find defects, but to analyze and recommend corrections. To do this they must develop a deep understanding of the system and each unit of code. This has a very positive effect on overall software quality: assuring that the code is understandable and maintainable. Since individuals on the team rotate through this role, they learn each other's parts of the system and transfer best programming practices among themselves. Another benefit is that programmers learn how to prevent common defects from being introduced.

- Infrastructure engineer.

  This individual is responsible for sustaining the hardware and software resources used by the development team. Like the other supporting roles, this is a proactive function. The infrastructure engineer not only reacts to problems, but anticipates them and works to keep things running smoothly. The infrastructure engineer must observe how the development system is used by the team and suggests and implements improvements that will increase developer productivity. An example of this activity is monitoring growth in utilization of processor or network resources, identifying potential saturation or bottlenecks, and identifying and implementing actions to mitigate these impediments before they become a hindrance to the team.

- Knowledge/Domain Engineer.

  Individuals fulfilling this role are responsible for analyzing and building models that capture the commonalty and variability in both the business/problem domain and the technology domain. This information is used by the architect to arrange the system to accommodate anticipated changes. The commonality/variability not only refers to existing systems, but also relates to trends and future needs. Knowledge/domain engineers work with technology consultants and subject matter experts to establish the domain-specific taxonomy, lexicon, concepts, ontologies, or patterns necessary to organize and retrieve information stored by the system.

- Data enterers/validators.

  These individuals provide the resources to identify and load data into the system to support prototype demonstrations, testing, or initial product deliveries as needed. This role may be staffed by inexpensive yet enthusiastic talent (e.g., college students). The backup architect oversees these individuals.

- Technology consultants/subject matter experts.

  These individuals are the experts in a technology, application domain, product, etc. Their services are usually acquired from vendors or academia; they are available for periodic scheduled and impromptu meetings. These people provide guidance to the team on which technologies to apply and how best to integrate these technologies. They act as transfer agents for technical or business knowledge from the customer or vendor/academic community to the team members. They are a considerable source of expertise for the knowledge/domain engineers.

2.2 Concept Initialization/Resetting Activities

The purpose of the Concept Initialization/Resetting activities is to develop or refine an architectural template for the system and to establish the overall requirements and goals for the development effort. Priorities for these goals are set so that an iteration of development (e.g., a time box) can be performed. In highly exploratory situations, the goals may deal with identifying user needs or investigating and providing better information about available implementation alternatives. For mature products, the goals could address delivery of operational capability to the end users. Figure 7 shows the Concept Initialization/Resetting Activities.

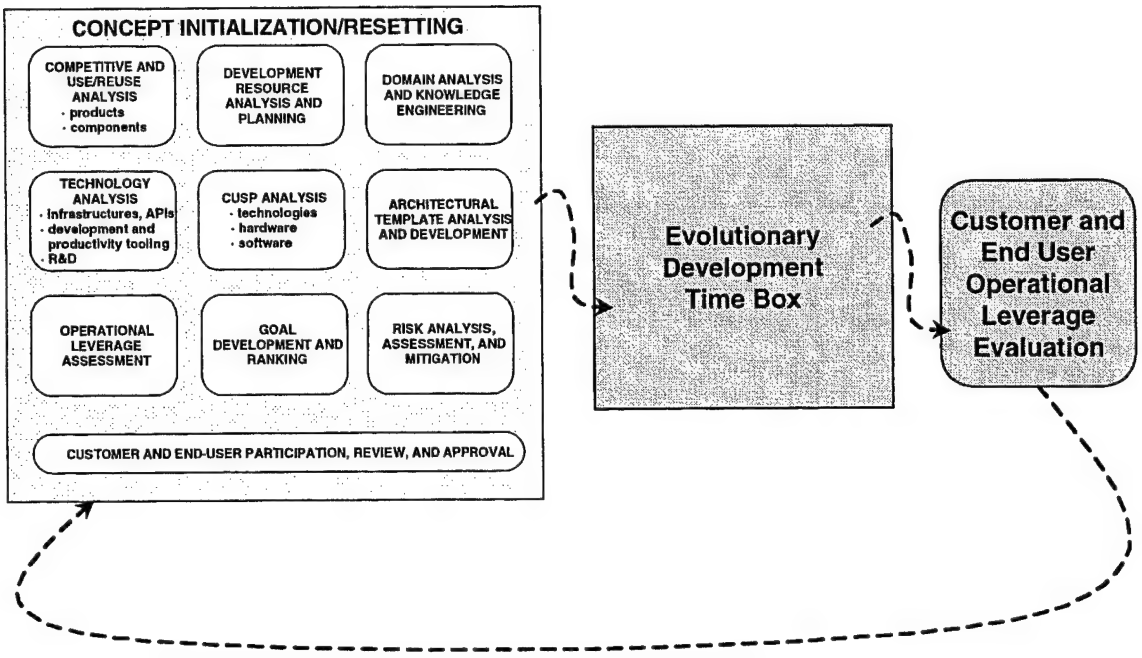

Figure 7. Concept Initialization/Resetting Activities

To initiate the Evolutionary Development Time Box, it is necessary to establish an architectural concept. Promising technologies and their implementations must be investigated, development environments identified, and their procurement planned. The proposed approach or alternatives are evaluated from the customer/user perspective, and overall risk is identified and assessed. Trends in the technology, business environment, hardware, and software are also factored into the analysis and concept setting. In particular, an analysis of the business domain, as well as the wider intersecting domains, are performed to capture the essential characteristics of systems within those domains.

The activities performed in this part of the ERD process represent many viewpoints into the major issues that affect producing a system. The team works together to integrate these viewpoints, making tradeoffs as necessary to achieve an effective balance. This includes both system- and software-specific issues. The team works on articulating a feasible solution to several interrelated issues. The team tries to eliminate sub-optimal approaches early and picks a strategy that has the greatest likelihood of satisfying the cost, schedule, quality, functionality, performance, and other objectives for the product.

The nine activities that are performed during this phase are elaborated in the following paragraphs.

2.2.1 Competitive and Use/Reuse Analysis

The purpose of this activity is to identify candidate COTS packages and assess their maturity and usefulness in implementing the system. Not only are the features of the product to be investigated, but assumptions about its internal structure and implementation must be uncovered to identify the product's operational limits. Often, the only way to mitigate reuse risk is to "fly before you buy," with emphasis on stress testing the item to understand its internal capabilities and limitations.

During the concept setting/resetting stage of the ERD process, this activity is a "quick-look"—separating the candidates that are feasible from those that are not. The team is primarily interested in assessing overall product maturity and identifying the features and feasibility of using the packages. Later, in the Exploratory Development Time Box, more detailed exploration will be performed as integration and performance issues are investigated. This activity interacts heavily with the architecture and technology analysis activities, as well as the Cusp Analysis activities. In some cases you may be evaluating early release (beta) software and have to make decisions based on planned changes for the products. The analysis furnished by this activity helps to determine the activities in the time box that should be conducted either to obtain information about the products or to develop strategies for encapsulating them so multiple products can be substituted if necessary.

2.2.2 Architectural Template Analysis and Development

This activity takes as input the analysis provided by the other activities to synthesize an architecture for a family of systems within the domain of interest. This template guides all development and integration efforts. The architectural template identifies the services and protocols used, stressing scalability, evolvability, and rapid and cost-effective accommodation of changes in the business or technical domains. The template documents the assumptions made about an instantiation of the architecture. It indicates how implementation decisions are to be isolated and encapsulated—for example, using virtual machines, server, or clients. A functional framework identifying user features is produced.

Design tradeoffs are one of the principal tasks performed in this activity. As implementations of components, interfaces, protocols, and standards are identified, the architecture is adjusted to allow for these to be included in a controlled manner. Often this adjustment requires adding or redefining the abstractions the architecture provides—for example, allowing for multiple search engines to be included, yet providing a consistent search service to the clients.

2.2.3 Technology Analysis

Whereas the Competitive and Use/Reuse Analysis activity (see Section 2.1.1) looks at implementations of technology, technology analysis activity looks at the basic technology itself. An example is the Object Request Broker (ORB) technology, which has been implemented by many vendor products (e.g., IBM SOM, DEC ObjectBroker, IONA Orbix). Analysis conducted in the Technology Analysis activity identifies the strengths and weaknesses/limitations of each technology (e.g., OLE/COM versus CORBA ORB), while the Competitive Use/Reuse analysis provides insight into maturity of each implementation (e.g., performance or scale-up).

Technology analysis examines standardization efforts to identifying those technologies that are in decline, mainstream, or emerging. Of particular interest is identifying alternative technologies so that a competitive use/reuse analysis of interoperability can be performed—for example, OLE/COM/DCOM interoperability/integration with CORBA technology via bridges in IONA Orbix.

2.2.4 Goal Development and Ranking

The Concept Initialization Setting/Resetting activities support developing achievable goals for the next iteration of the time box. The goals may be multidimensional, focused on obtaining information on a technology or its implementation. The goals may also be specific to implementing a capability (e.g., features) for delivery to the users. All stakeholders have a responsibility to assure that attainable and realistic goals are established. Setting attainable goals is tightly coupled with the risk analysis. The risks related to attaining a goal within the constraints must be known and agreed upon by all the stakeholders.

Goals (such as feature lists) are ranked to assure that at the end of a time box, a tangible product is delivered. The ranking helps to distinguish between those features that are essential to meet time box deadlines and those that can be delayed.

2.2.5 Risk Analysis and Mitigation

As the goals for the system are defined, risks related to achieving those goals are identified. Actions are planned either to reduce the chance of failure or to limit its impact. In some cases the strategy for dealing with a risk influences the goals for the time box, possibly focusing action on providing better and more reliable information on user need or implementation. Early in the life cycle, we explicitly plan time to allow for investigating alternatives and eliminating unproductive approaches from consideration. The objective is to leave as many options open for as long as possible. This flexibility is a consideration in defining and revising the system architecture.

2.2.6 Domain Analysis and Domain Engineering

The purpose of this activity is to identify underlying themes in the business and technology domains, and to provide this information to the system architect to help encapsulate decisions regarding the variability and similarity of systems in the domain. This helps to identify biases or implementation assumptions embedded in the way people currently work and to allow for the analysts to get at the essence of the problem. Domain-specific attributes are isolated to a knowledge base. This analysis not only supports the investigation of a narrow domain (e.g., word processing), but addresses the wider environments of interest in which the domain operates or interacts (e.g., word processing as part of an overall office suite). This analysis supports developing the domain specific taxonomies, lexicons, concepts, ontologies, and patterns for information to be stored and retrieved from the system. This analysis relies on the support of technology consultants and subject matter experts.

2.2.7 Cusp Analysis

Cusp analysis is at the center of the Concept Initialization/Resetting activities since it affects decisions made in the other activities. The analysis gauges change in the underlying technologies, components, tools, user skills/knowledge, and usage environment, and focuses on predicting how these changes will affect the architectural and implementation decisions for the system. This activity is focused on anticipating trends and establishing strategies to converge with advances in implementations of technologies. Factors evaluated include the rate of change (or stability) of products, price/performance, and publication and community acceptance of standards.

Cusp analysis builds a strategy to guide both the packaging of functionality that can be delivered now and identifying what to delay. The emphasis is on allowing the system to adapt in an orderly way with minimum rework and to leverage new COTS products. This forecasting extends the useful life of the system and helps reduce development costs either by avoiding needless implementation functionality that may soon be available, or by implementing hooks that will allow for it to be incorporated later. An example is the rapid shift to the Internet as a means of conducting business and increased reliance on remote collaboration. Another example is the shift from Windows 3.x (16-bit) platforms to Windows 95 (32-bit) platforms and the emergence of OCX/ActiveX technology.

2.2.8 Operational Leverage Assessment

This activity provides a focus for understanding and articulating the business needs and constraints that the system must satisfy for the user/customer to get the expected value. If the system cannot offer appreciable strategic or tactical/operational value, it may not be worth pursuing development at this time. This analysis is life cycle-oriented; that is, it helps rank the functionality that should be developed early versus that which can be delayed. The Operational Leverage Assessment is a basis of the system's evolutionary growth plan and is used to determine goals for each Exploratory Development Time Box. In addition to functionality, the Operational Leverage Assessment considers operational constraints such as availability, reliability, and maintainability.

2.2.9 Development Resource Analysis and Planning

As the overall concept for the system is established, the technical resources necessary to implement the system must be determined. This includes identifying and ordering development platforms (e.g., workstations/servers with tools) as well as integration platforms (i.e., typical user workstations: clean machines for integration). Agreements for access to beta releases of software are initiated. Information resources such as subscription services and knowledge bases (e.g., Microsoft Developers Network) are identified and procured. Technical and subject matter experts are also identified and their services negotiated at this time.

The maturity of the development resources provides input for decisions to adopt and build the system using an emerging technology. If the development kits are immature, then it is possible to delay or restrict the use of the technology. Cusp analysis helps develop projections for when to adopt the technology into the project.

2.3 Exploratory Development Time Box Activities

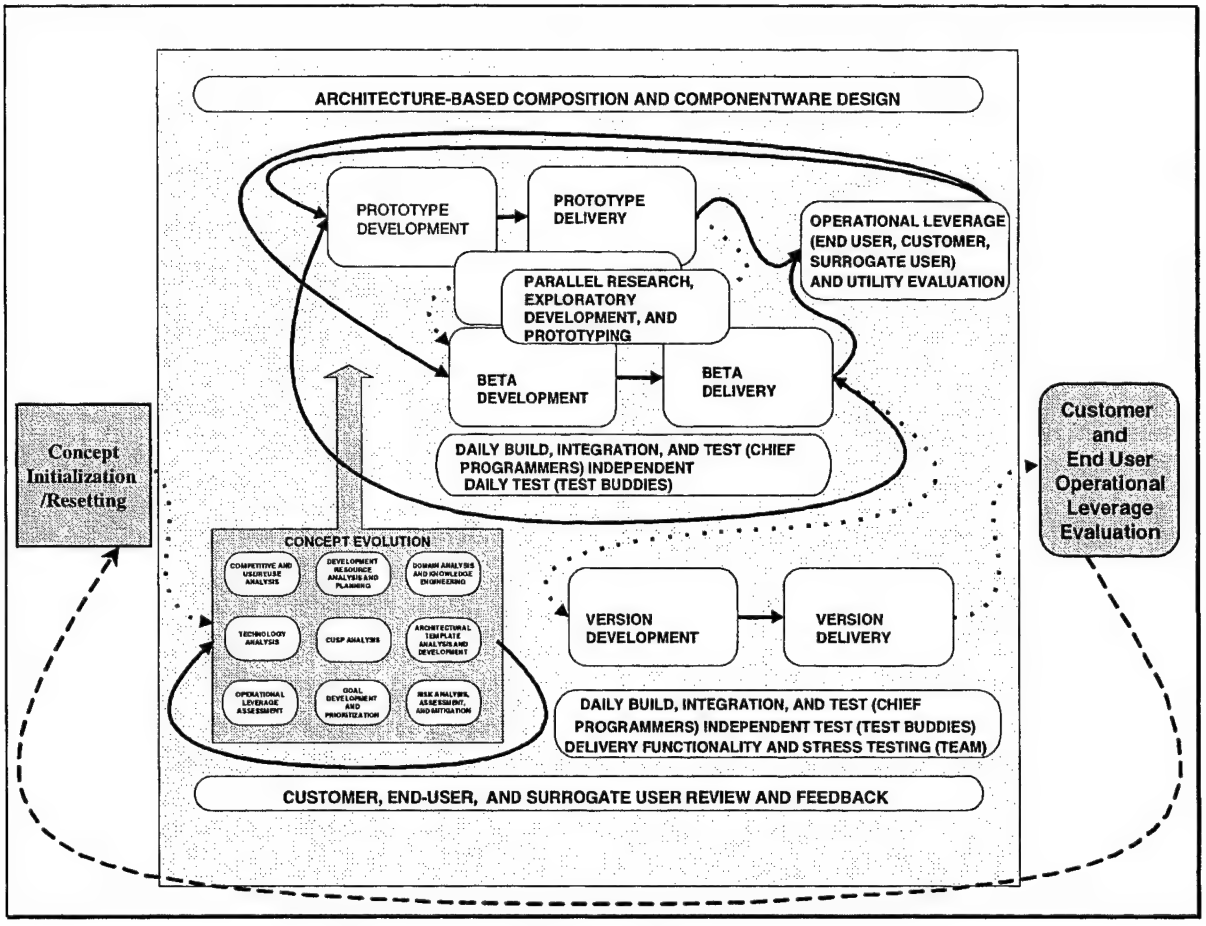

Figure 8.

The Exploratory Development Time Box activities of the ERD process apply effort to evolve the system to its next level. As mentioned in Section 2.2, the goals for the time box are established by the Concept Initialization/Resetting activities. The time box uses that analysis as its basis for refining the system. The time box is scheduled to execute within a specific time period with a specific set of resources. More than one time box may be underway at any given moment. Time boxes may also be recursive, spawning off multiple, separately managed time boxes when necessary (e.g., a need to investigate several alternatives in parallel). Three principal types of development activity may take place in a time box as described below:

- Prototype development.

  Prototypes may be of the "throw-away" variety, or they can serve as the basis for developing beta products. Throw-away prototypes are used to obtain reliable information on implementation approaches or to solicit user requirements. Prototype products are not necessarily integrated; they may consist of a loose collection of programs or mockups. User/customer (surrogate or actual) feedback sessions provide insight so that adequate

understanding of needs can be obtained. These needs, along with some of the prototypes, are then used to develop beta versions.

- Beta development.

  These betas integrate functionality for all or part of a user process. Like prototypes they are evaluated by (surrogate or actual) users. These beta versions have most of the functional characteristics of a system, but they may not have all the attributes necessary for full deployment in actual operational situations (e.g., performance, scale-up). Beta systems generally operate under controlled circumstances and have known bugs and workarounds. They may have been adapted to overcome difficulties in implementing the architecture (e.g., some components may not be available or not operating as expected). Beta systems are evaluated to determine their impact on the business process and to assess operational leverage with realistic usage scenarios.

- Version development.

  Fully functioning, production-quality systems are the focus of this activity. The system undergoes its most comprehensive test and evaluation by the development team and is prepared to be released and maintained in the customer's operational environment. Bugs may be tolerated in the final version as long as they are clearly identified, their effect isolated, and plans are readied to remove the defects in future versions. The quality goals for the product are defined as part of time-box planning.

Within each time box, the concept is allowed to evolve by revising the analysis generated from the Concept Initialization/Resetting activities. The concept is refined to reflect decisions made about both the features that will be implemented and the use of components. The operational leverage and usability of prototypes and beta deliveries are assessed and used to determine if the business goals can be met. Minor adjustments to the architecture and other analysis products are tolerated. Significant changes that may violate the constraints of the time box (e.g., the goals are no longer attainable) requires that the time box be terminated and new Concept Setting/Resetting activities be initiated.

In addition to customer and user feedback, a review and test process is used to measure and improve overall quality of the intermediate (e.g., prototype and beta deliveries) and final version. Code reviews are conducted frequently by the architect and lead design (or backup) as well as the chief programmers, testers, and documentation support persons. The code must be able to communicate its intentions clearly with minimal need for external commentary. The coding style (e.g., aesthetics) must be consistent across the team to support readable and understandable code. Testing relies on a thorough knowledge of the code. As the system reaches "critical mass," daily integration and testing (including stress testing) is conducted.

2.4 Customer and End-User Operational Leverage Evaluation Activities

The customer and end-user operational leverage evaluation is a formal review and acceptance of the developed product by the customer and user community. Once accepted, the product is installed in its operational environment and supported. If the system does not meet current operational needs, then another Concept Initialization/Resetting step is performed. This is sometimes necessary since the underlying philosophy of time-box development is to meet minimal development goals by a fixed deadline. Features that may be needed by the end users may be lacking, or the user's need may have shifted from when the need was first articulated. Another time-box session will be scheduled to include these needs as essential features.

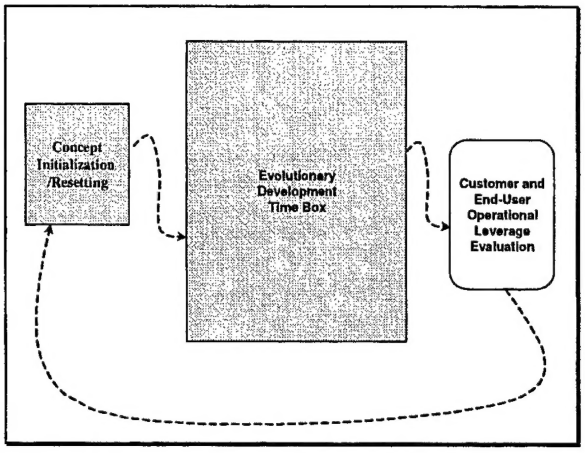

Figure 9.

3.0 Experiences

Some of our more significant lessons learned from developing and applying the ERD process are described below:

- Transitioning to a defined process.

  When the Advanced Development Products team first began the CYBERosetta development efforts, we relied almost exclusively on our most capable and experienced team members to guide the development decisions. The ERD process described here is an attempt to articulate, after the fact, our lessons learned from what we did. We have combined the essential actions into a set of activities, and we are currently using these descriptions to guide future development efforts. As we continue to use the process, we continue to refine its description and continue to incorporate the lessons learned to make the ERD process more repeatable.

- Systems development is about learning.

  If you are not failing, you are not extending your horizons, and you are not learning. You obtain more information about the nature of a solution/approach when you push the envelope. The trick is to get experience early and build on successes, rather than to build on assumptions that turn out to be proven false later in development. We have learned to anticipate when we may have to throw away some approaches, but we retain the rationale and experiences.

  It is important to recognize that not all efforts will make it into the final product. Developers do not readily accept the fact that they cannot make something work. We

have learned to build a culture where capturing the experience and knowledge is just as important. This approach is especially helpful during prototyping activities. It is important to communicate to developers when something is risky, that alternative approaches may be tried, that their product is a means to evaluate an issue, and that that product may be discarded. This helps team morale; no one likes to assume they have "failed" and to have their work products discarded. If something looks like it is unfruitful, you must be able to abandon it before you consume resources unproductively.

- Invest in acquiring new skills/knowledge.

  The pace of technological change is intimidating; you have to run just to stay in the same place. No longer can we be casual about acquiring new skills and information. The ERD process relies on leveraging commercially available products using standards-based architectures. We must be aggressive in tracking emerging technologies and commercial implementations. This requires resources and commitment from all team members. No matter what the personal commitment, the time to invest in pursing this information must be made available. That is why we explicitly program into the development resources some time (currently 10%) that team members *must* devote to investigating and learning about new products and technologies. We have begun to assign technology areas (e.g., collaboration technology) to individuals as a way to assure they develop necessary expertise. Team members have the responsibility to transfer their insight to one another.

- Product/process documentation.

  The development process concerns making and communicating decisions and their rationale in real time to all stakeholders (this includes developers, marketing, senior management, and customers). It is necessary to record these critical decisions so that everyone is on the "same sheet of music" and can operate effectively as a team. Since the architecture/design is central to the way we develop systems, we are evolving architectural diagrams and descriptions that communicate the essence of the system and that serve as the basis for organizing its implementation. We are experimenting with techniques that make obtaining and recording design decisions ubiquitous and natural, rather than a separately orchestrated documentation process. For example, we are examining computer-connected electronic white boards and video conferencing as a means to capture architectural and design decisions and rational.

- Code review.

  As productivity increases, especially using senior programmers/analysts, a large amount of code is developed over short periods of time. To review this code, the ratio of architect to designers appears to be about 1 to 4. It is critical, therefore, to establish a backup for the architect/lead designer early in the life cycle to help share in review responsibilities.

- Administrative support.

  The administration function keeps the process moving smoothly. In a process built on rapid application development fundamentals, events happen quickly. The administrative

function must be closely integrated with the team to provide a planning and oversight function while assuring that organizational needs for visibility into the project are met. Integrated support tools that help exchange accurate and timely cost/schedule data between the parent organization's management information system and the project's processes is essential.

- Hardware/software infrastructure.

  To enable rapid, or even parallel, development of functionality, significant hardware and software resources are needed. We cannot allow our most critical resource (the developers) to be bottlenecked and constrained by infrastructure. Serious consideration must be made to upgrade and obtain productivity and development tool enhancements throughout the life cycle. A development environment, once established, must be continually revised to meet developer needs.

- Customer/user interaction essential.

  No matter how well we think we understand our customer or user needs, nothing is comparable to having a fresh perspective from outside the development team. We rely on frequent and often impromptu feedback from our user community to give us insights that we do not have the ability to generate ourselves. Having a development approach that stresses frequent integration and testing facilitates the ability to have the user community provide this insight.

4.0 Next Steps

As we continue to gain experience with the ERD process, we will refine the process steps and techniques and record additional guidance on how to perform the activities. The following topics are currently candidates for investigation:

- Process measurements.

  We currently rely on experienced individuals to predict our effort, cost, and schedule parameters when planning a time box. We are exploring how to quantify this knowledge and develop metrics to plan and manage the project. In particular we wish to investigate the relationship of delivered capability to changes in operational leverage, leading to an estimate of return on investment.

- Relationship to current process frameworks.

  The ERD process grew out of our need to manage and direct highly-motivated individuals as they define and deliver advanced information systems built on commercial technology. Towards this end the process has achieved its objective. It has provided us with the right amount of structure to deal with the important issues we face. We would like to improve our processes, and hence we are looking at the S/W Capability Maturity Model, ISO 9000, and the Systems Engineering CMM to provide insight into additional practices that we can incorporate into our process.

- Additional guidelines.

  As we formalize the process descriptions, we must capture our knowledge of and experience with the process so that others performing the activities can benefit from our lessons learned. We will establish mechanisms to capture relevant experience and communicate it throughout the team. Some areas on which we will focus are guidance on performing the assessment and analysis activities such as Use/Reuse, Technology, and Cusp Analysis. We will also create checklists for performing process activities consistently, such as code reviews and prototype evaluations.

References

Booch, Grady. 1997. Components, Continuously Changing Systems, and Urban Sprawl, Object Magazine, April.

Cusumano, Michael A. and Richard W. Selby. 1995. Microsoft Secrets, The Free Press.

Jones, Capers. 1995. End-User Programming, IEEE Computer, September.

Lewis, Ted. 1995. Living in real time, side A (What is the Info Age), IEEE Computer, September.

McConnel, Steve. 1996. Rapid Development, Microsoft Press.

O'Leary, Daniel E. 1997. The Internet, Intranets, and the AI Renaissance, IEEE Computer, January.